```
# -*- coding: utf-8 -*-
"""
Created on Mon Mar 25 14:16:41 2019

@author: Nate
"""

import numpy as np
import matplotlib.pyplot as plt
import pdb



delta_r = .01
l_orbital = [0,1,2,3]
#l_orbital = [0]
eps_list = []

#-1/r potential energies
#for i in range(1001):
#    eps_list.append(i*.001-1)


#r*r/2 potential energies
for i in range(1001):
    eps_list.append(i*.01)

#u(r-dt), u(r), l, r, eps
def stepping(u_2, u_1, l, r, eps):
    fun = 2*(r*r/2) + l*(l+1)/r**2 - 2*eps
    #fun = -2/r + l*(l+1)/r**2 - 2*eps
    u_3 = 2*u_1 - u_2 + delta_r**2*fun*u_1
    return(u_3)



#################################################################
#################################################################


good_points = []

for l in l_orbital:
    for eps in eps_list:
        u = [0, .01]

# -*- coding: utf-8 -*-
"""
Created on Tue Mar 26 19:46:14 2019

@author: Nate
"""

import numpy as np
import matplotlib.pyplot as plt
import pdb


'''
Determine the value of π to ≈ 14 digits by solving for the root of the equation

f(x) = cos(x) = 0

using the second order Newton’s method. The exact solution is

    x∗ = π/2, so that π = 2x∗.

Use the initial guess of x = 1.5, corresponds to a guess of π ≈ 3. How many
iterations are needed to achieve 14 digits? Repeat the calculation with initial
guesses
x = 1, x = 0.5 and x = 0.25.
'''
```

```
        r = delta_r*2


        for i in range(5000):
            u.append(stepping(u[-2],u[-1], l, r, eps))

            if u[-1]*u[-2] < 0:
                good_points.append([l, eps, r])

            r+=delta_r
            u = [u[-2],u[-1]]


############################################################
############################################################
#Plotting


l0 = []
l1 = []
l2 = []
l3 = []

for i in range(len(good_points)):
    if good_points[i][0] == 0:
        l0.append([good_points[i][1],good_points[i][2]])
    elif good_points[i][0] == 1:
        l1.append([good_points[i][1],good_points[i][2]])
    elif good_points[i][0] == 2:
        l2.append([good_points[i][1],good_points[i][2]])
    elif good_points[i][0] == 3:
        l3.append([good_points[i][1],good_points[i][2]])


fig1, axes1 = plt.subplots()
axes1.scatter([l0[i][1] for i in range(len(l0))],[l0[i][0] for i in range(len(l0))])
axes1.scatter([l1[i][1] for i in range(len(l1))],[l1[i][0] for i in range(len(l1))])
axes1.scatter([l2[i][1] for i in range(len(l2))],[l2[i][0] for i in range(len(l2))])
axes1.scatter([l3[i][1] for i in range(len(l3))],[l3[i][0] for i in range(len(l3))])
axes1.set_ylabel('Energy')
axes1.set_xlabel('r')
axes1.set_title("Energy Orbitals", va='bottom')
axes1.legend(('l=0','l=1','l=2', 'l=3'), loc='upper right')
plt.show()

def f(guess):
    return(np.cos(guess))

def df(guess):
    return(-np.sin(guess))



guess = [1.5,1,.5,.25]


for i in range(len(guess)):
    for val in range(4):
        nextguess = guess[i] - f(guess[i])/df(guess[i])
        guess[i] = nextguess

    if i == 3:
        answer = guess[i]
        #print(guess[i])
        #print(answer)
        print(i, ": " , (3/2)*np.pi-answer)
        if abs((3/2)*np.pi-answer) < 10**(-12):
            print('True')
    else:
        answer = 2*guess[i]
        print(i, ": " , np.pi-answer)
        if abs(np.pi-answer) < 10**(-12):
            print('True')
```

```
# -*- coding: utf-8 -*-
"""
Created on Thu Mar 28 15:38:36 2019

@author: Nate
"""


import numpy as np
import matplotlib.pyplot as plt
import pdb

'''
Use the Killingbeck method as presented in class to solve for the eigenvalues of the
hydrogen atom as in 1), but with greater accuaracy. Use the same Verlet algorithm
to integrate backward to the origin from r = 25-50. Do Newton's iterations 4-10
times to find the correct e so that u(0,e) = 0. Determine the lowest energy levels of
l = 0, 1, 2, 3. Use e = −0.6 as your initial guess energy. Plot all energy values as a
function of dr from 0.01 to 0.1.
'''

delta_r = []
l_orbital = [0,1,2,3]
#l_orbital = [0]
energy_guess = -.6

#-1/r potential energies
for i in range(1,11):
#for i in range(1,50):
    delta_r.append(i*.01)

#u(r+dr), u(r), l, r, energy_guess
def stepping(u_r_plus_dr, u_r, l, r, eps):
    #fun = 2*(r*r/2) + l*(l+1)/r**2 - 2*eps
    #pdb.set_trace()
    fun = -2/r + l*(l+1)/r**2 - 2*eps
    u_r_minus_dr = 2*u_r - u_r_plus_dr + delt_r**2*fun*u_r
    return(u_r_minus_dr)

def stepping_for_v(v_r_plus_dr, v_r, u_r, l, r, eps):
    #fun = 2*(r*r/2) + l*(l+1)/r**2 - 2*eps
    #pdb.set_trace()
    fun = -2/r + l*(l+1)/r**2 - 2*eps
    v_r_minus_dr = 2*v_r - v_r_plus_dr + delt_r**2*fun*v_r+ delt_r**2*-2*u_r
    return(v_r_minus_dr)


#################################################################
#################################################################


good_points = []
to_plot = []


for l in l_orbital:
    to_plot_holder = []
    for delt_r in delta_r:
        r = 100 - delt_r*2

        for it in range(20):  #8
            u = [0, .01]  #u_r_plus_dr, u_r
            v = [0, .01]
            #print(int(round(r/delt_r)))


            for i in range(int(round(r/delt_r))):

                u[0],u[1] = u[1], stepping(u[-2],u[-1], l, r, energy_guess)
                v[0],v[1] = v[1], stepping_for_v(v[-2],v[-1], u[0], l, r, energy_guess)
                #u.append(stepping(u[-2],u[-1], l, r, energy_guess))

                if u[0]*u[1] < 0:
                    good_points.append([l, energy_guess, r])

                r-=delt_r
                #u = [u[-2],u[-1]]
            energy_guess = energy_guess - u[0]/(v[0]+.000001)

#        print("delta r is: ", delt_r, "  Energy is: ",energy_guess)
        to_plot_holder.append([delt_r, energy_guess])
    to_plot.append(to_plot_holder)


#################################################################
#################################################################
#Plotting



l0 = []
l1 = []
l2 = []
l3 = []


for i in range(len(good_points)):
    if good_points[i][0] == 0:
        l0.append([good_points[i][1],good_points[i][2]])
    elif good_points[i][0] == 1:
        l1.append([good_points[i][1],good_points[i][2]])
    elif good_points[i][0] == 2:
        l2.append([good_points[i][1],good_points[i][2]])
    elif good_points[i][0] == 3:
        l3.append([good_points[i][1],good_points[i][2]])


fig1, axes1 = plt.subplots()

axes1.scatter([to_plot[0][i][0] for i in range(len(to_plot[0]))], [to_plot[0][i][1] for i in range(len(to_plot[0]))])
axes1.scatter([to_plot[1][i][0] for i in range(len(to_plot[1]))], [to_plot[1][i][1] for i in range(len(to_plot[1]))])
axes1.scatter([to_plot[2][i][0] for i in range(len(to_plot[2]))], [to_plot[2][i][1] for i in range(len(to_plot[2]))])
axes1.scatter([to_plot[3][i][0] for i in range(len(to_plot[3]))], [to_plot[3][i][1] for i in range(len(to_plot[3]))])
axes1.set_ylabel('Energy')
axes1.set_xlabel('$\Delta$r')
axes1.set_title("Energy as a Function of $\Delta$r", va='bottom')
axes1.legend(('l=0','l=1','l=2', 'l=3'), loc='upper right')
#plt.show()
```